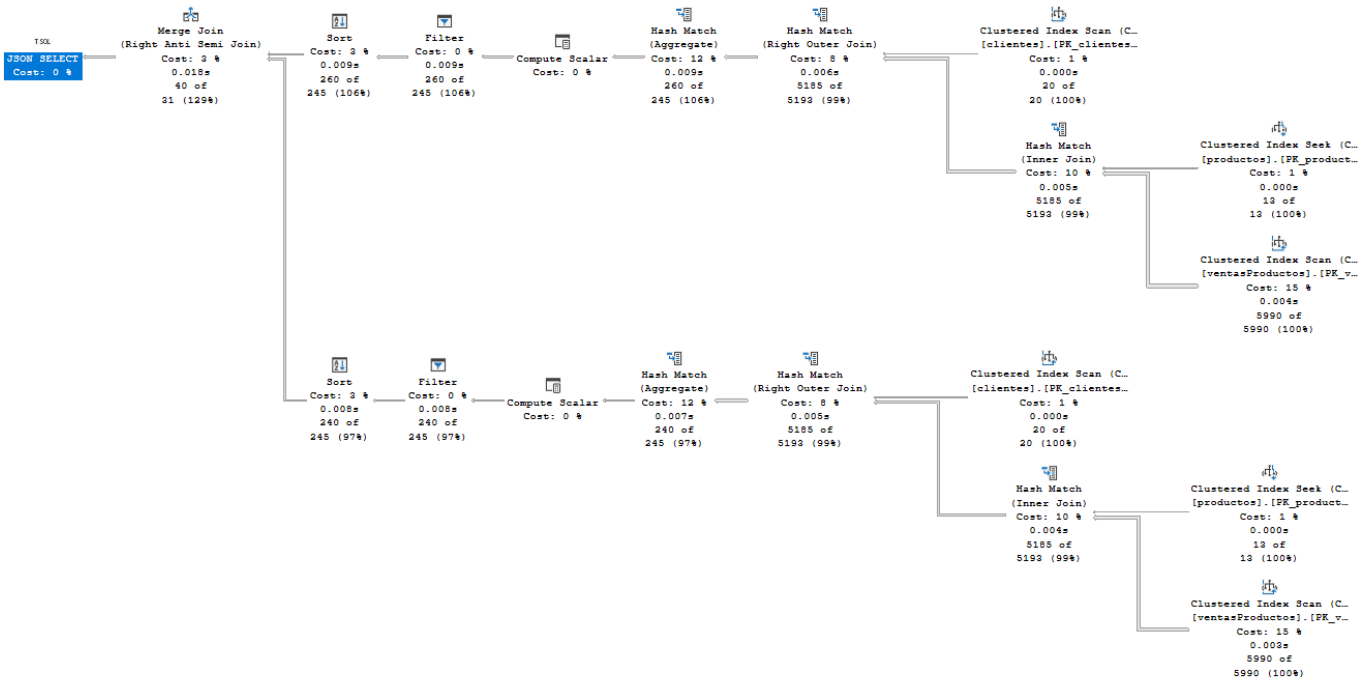
Consulta Original Punto 3:

```
SELECT
    p.nombre AS producto,
    c.nombre AS cliente,
    COUNT(v.ventaid) AS cantidad_ventas,
    SUM(v.cantidadVenta) AS total_ventas
FROM
    ventasProductos v
    INNER JOIN productos p ON v.productoid = p.productoid
    LEFT JOIN clientes c ON v.clienteid = c.clienteid
WHERE
    p.productoid NOT IN (2, 5)
    AND v.cantidadVenta >2
GROUP BY
    p.nombre,
    c.nombre
HAVING
    COUNT(v.ventaid) >= 2
EXCEPT
SELECT
    p.nombre AS producto,
    c.nombre AS cliente,
    COUNT(v.ventaid) AS cantidad_ventas,
    SUM(v.cantidadVenta) AS total_ventas
FROM
    ventasProductos v
    INNER JOIN productos p ON v.productoid = p.productoid
    LEFT JOIN clientes c ON v.clienteid = c.clienteid
WHERE
    p.productoid NOT IN (3, 6)
    AND v.cantidadVenta > 2
GROUP BY
    p.nombre,
    c.nombre
HAVING
    COUNT(v.ventaid) >= 2
ORDER BY
    total_ventas DESC
FOR JSON AUTO
```

Funcionamiento de la consulta:
Se seleccionan los campos nombre de la tabla productos, nombre de la tabla clientes, cantidad_ventas (utilizando la función COUNT) y total_ventas (utilizando la función SUM) de la tabla ventas.
Se hace un JOIN interno entre las tablas ventas y productos utilizando la columna productoid.
Se hace un LEFT JOIN entre las tablas ventas y clientes utilizando la columna clienteid.
Se aplican condiciones en la cláusula WHERE para seleccionar solo las ventas de productos cuyo productoid no sea igual a 2 o 5, y cuyo monto sea mayor a 2.
Se agrupan los resultados por nombre de productos y nombre de clientes.
Se aplica una condición en la cláusula HAVING para seleccionar solo los resultados que tengan al menos 2 ventas (cantidad_ventas).
Se utiliza el operador EXCEPT para excluir los resultados que incluyan productos con productoid igual a 3 o 6.

Se agrupan los resultados nuevamente por nombre de productos, nombre de clientes, cantidad_ventas y total_ventas.
Se ordenan los resultados por cantidad_ventas y total_ventas de forma descendente.
Se utiliza la cláusula FOR JSON para obtener los resultados en formato JSON.

Plan de ejecución original:



Normas de Optimización

Norma 1:

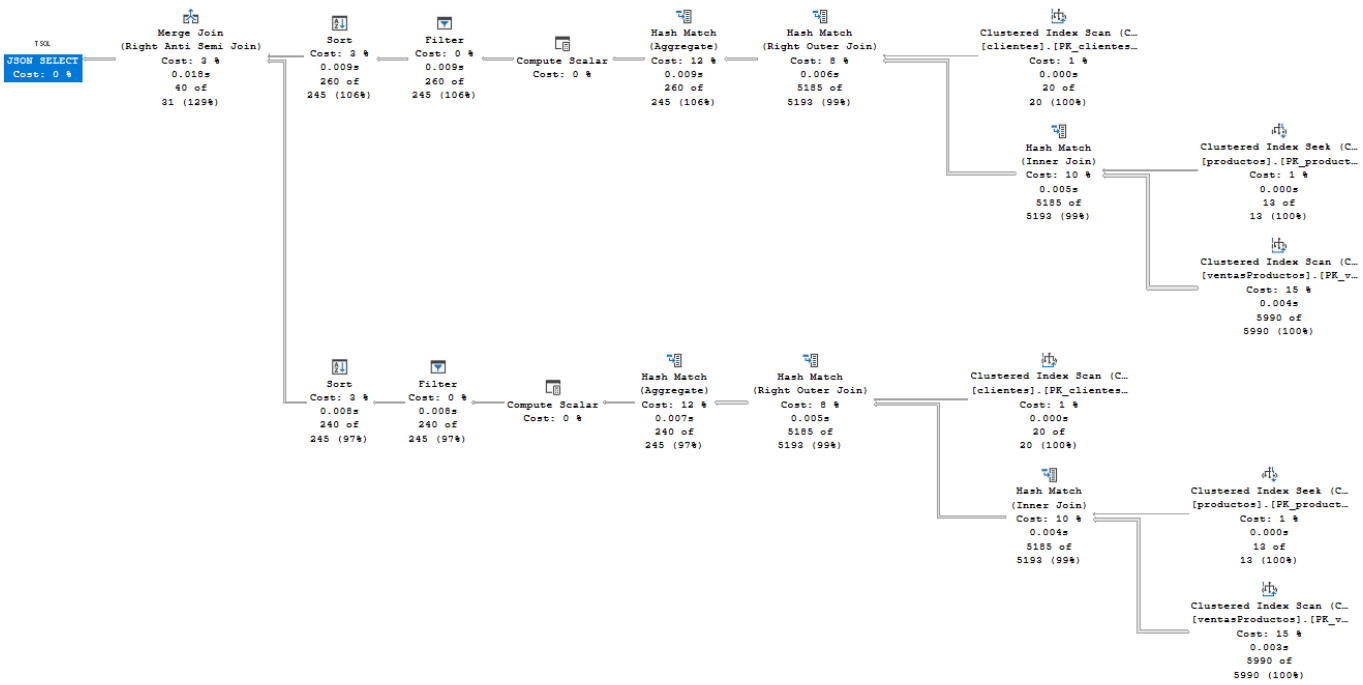
Unidad de workload	Explicación	Norma
Clustered Index Scan (Clustered) Scanning a clustered index, entirely or only a range. **Physical Operation**: Clustered Index Scan **Logical Operation**: Clustered Index Scan **Actual Execution Mode**: Row **Estimated Execution Mode**: Row **Storage**: RowStore **Actual Number of Rows Read**: 10090 **Actual Number of Rows for All Executions**: 5990 **Actual Number of Batches**: 0 **Estimated I/O Cost**: 0.0697917 **Estimated Operator Cost**: 0.0810477 (15%) **Estimated Subtree Cost**: 0.0810477 **Estimated CPU Cost**: 0.011256 **Estimated Number of Executions**: 1 **Number of Executions**: 1 **Estimated Number of Rows for All Executions**: 5990 **Estimated Number of Rows Per Execution**: 5990 **Estimated Number of Rows to be Read**: 10090 **Estimated Row Size**: 20 B **Actual Rebinds**: 0 **Actual Rewinds**: 0 **Ordered**: False **Node ID**: 9 **Predicate** [elementalGDB].[dbo].[ventasProductos].[cantidadVenta] as [v].[cantidadVenta]>(2.00) **Object** [elementalGDB].[dbo].[ventasProductos].[PK_ventasProductos] [v] **Output List** [elementalGDB].[dbo].[ventasProductos].clienteId, [elementalGDB].[dbo].[ventasProductos].productoId, [elementalGDB].[dbo].[ventasProductos].cantidadVenta	Basándose en el predicado el motor de la base de datos se encarga de devolver el clienteid, el productoid, y la cantidadVenta donde en ventasProductos la cantidadVenta sea mayor a 2.	Creación de índices en las columnas ya que acelerará el tiempo de búsqueda al reducir la cantidad de filas que se deben escanear para filtrar los datos. Además, como los índices solo almacenan los valores de las columnas utilizadas, también se reduce el uso de memoria.

Aplicación de norma 1:

```
CREATE INDEX index_productoid ON productos (productoid);
CREATE INDEX index_monto ON ventasProductos (cantidadVenta);
CREATE INDEX index_clienteid ON clientes (clienteid);
CREATE INDEX index_ventas_productoid ON ventasProductos (productoid);
CREATE INDEX index_ventas_clienteid ON ventasProductos (clienteid);

SELECT
    p.nombre AS producto,
    c.nombre AS cliente,
    COUNT(v.ventaid) AS cantidad_ventas,
    SUM(v.cantidadVenta) AS total_ventas
FROM
    ventasProductos v
    INNER JOIN productos p ON v.productoid = p.productoid
    LEFT JOIN clientes c ON v.clienteid = c.clienteid
WHERE
    p.productoid NOT IN (2, 5)
    AND v.cantidadVenta >2
GROUP BY
    p.nombre,
    c.nombre
```

```
HAVING
    COUNT(v.ventaid) >= 2
EXCEPT
SELECT
    p.nombre AS producto,
    c.nombre AS cliente,
    COUNT(v.ventaid) AS cantidad_ventas,
    SUM(v.cantidadVenta) AS total_ventas
FROM
    ventasProductos v
    INNER JOIN productos p ON v.productoid = p.productoid
    LEFT JOIN clientes c ON v.clienteid = c.clienteid
WHERE
    p.productoid NOT IN (3, 6)
    AND v.cantidadVenta > 2
GROUP BY
    p.nombre,
    c.nombre
HAVING
    COUNT(v.ventaid) >= 2
ORDER BY
    total_ventas DESC
FOR JSON AUTO
```



Resultados:

Aplicando la norma 1 no se encuentra diferencia sustancial por lo que se seguirán buscando normas para optimizar.

Norma 2:

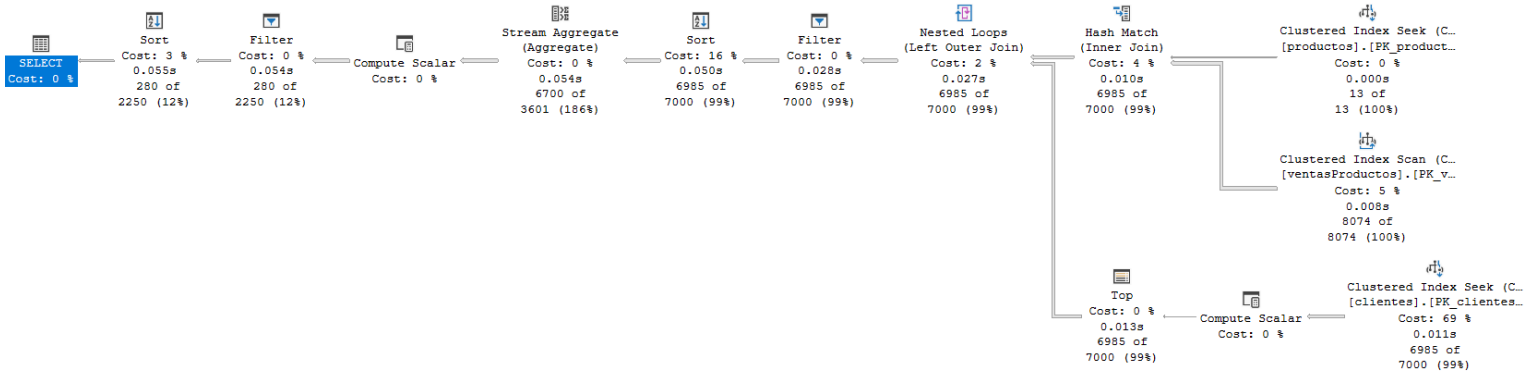
Se modifica la consulta de esta manera para generar optimizaciones:

```
SELECT
    p.nombre AS producto,
    c.nombre AS cliente,
    v.fechaVenta,
    COUNT(v.ventaid) AS cantidad_ventas,
    SUM(v.cantidadVenta) AS total_ventas
FROM
    ventasProductos v
    INNER JOIN productos p ON v.productoid = p.productoid
    OUTER APPLY (
        SELECT TOP 1
            c1.clienteid,
            c1.nombre
        FROM
            clientes c1
        WHERE
            c1.clienteid = v.clienteid
    ) c
WHERE
    p.productoid NOT IN (2, 5)
    AND v.cantidadVenta > 1
    AND c.clienteid IS NOT NULL
GROUP BY
    p.nombre,
    c.nombre,
    v.fechaVenta
HAVING
    COUNT(v.ventaid) >= 2
ORDER BY
    total_ventas DESC,
    cantidad_ventas DESC
```

En donde se aplicaron los siguientes pasos:

1. Utilizar la cláusula WHERE para filtrar los datos antes de unir las tablas. En lugar de utilizar la cláusula LEFT JOIN para unir la tabla clientes, podemos filtrar los datos por clienteid en la cláusula WHERE.
2. Utilizar la cláusula APPLY para unir la tabla clientes. La cláusula APPLY es más eficiente en algunos casos porque sólo une las filas que coinciden en lugar de unir todas las filas de la tabla.
3. Utilizar subconsultas en lugar de la cláusula EXCEPT para filtrar los resultados. Las subconsultas son más eficientes en algunos casos porque sólo seleccionan las filas que coinciden con los criterios de búsqueda.
4. Utilizar índices adecuados para las columnas utilizadas en las cláusulas JOIN, WHERE, GROUP BY y ORDER BY.
5. Utilizar la cláusula ORDER BY al final de la consulta, en lugar de utilizarla en el medio.
6. Utilizar la cláusula TOP para limitar el número de filas devueltas por la consulta.

Plan de ejecución optimizado:



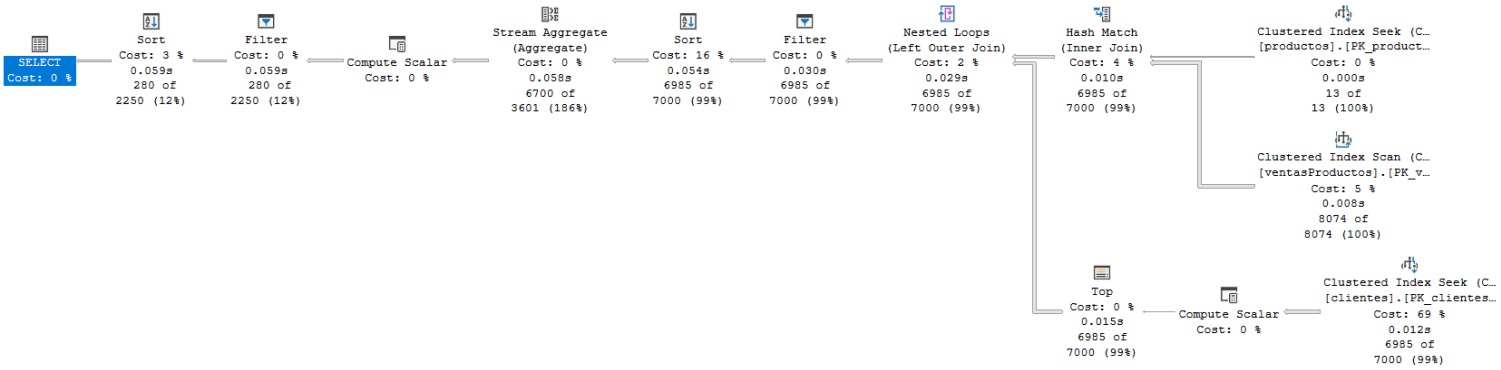
Resultados:

La consulta se optimiza de manera importante gracias a los cambios aplicados.

Aplicando cte:

```
WITH cteVentas AS (
    SELECT
        p.nombre AS producto,
        c.nombre AS cliente,
        v.fechaVenta,
        COUNT(v.ventaid) AS cantidad_ventas,
        SUM(v.cantidadVenta) AS total_ventas
    FROM
        ventasProductos v
        INNER JOIN productos p ON v.productoid = p.productoid
        OUTER APPLY (
            SELECT TOP 1
                c1.clienteid,
                c1.nombre
            FROM
                clientes c1
            WHERE
                c1.clienteid = v.clienteid
        ) c
    WHERE
        p.productoid NOT IN (2, 5)
        AND v.cantidadVenta > 1
        AND c.clienteid IS NOT NULL
    GROUP BY
        p.nombre,
        c.nombre,
        v.fechaVenta
    HAVING
        COUNT(v.ventaid) >= 2
```

```
)
SELECT *
FROM cteVentas
ORDER BY
    total_ventas DESC,
    cantidad_ventas DESC;
```



Resultados:

En la consulta original, se utilizaron subconsultas y joins en línea para obtener la información de los clientes, lo que puede ser menos eficiente que utilizar un CTE, sin embargo, al revisar plan de ejecución podemos ver que no cambia en nada particular, por lo que el equipo podrá decidir si usar la consulta original, o utilizar el CTE.

Una de las ventajas, que no tiene que ver con rendimiento, es el orden en que el CTE acomoda la consulta, pues podría ser más fácil de entender para el equipo.